SHEF Decoder Operations Guide

National Weather Service
Office Of Hydrologic Development
December 27, 2002

Table of Contents

1. INTRODUCTION

The SHEF Decoder is the primary means by which hydrometeorologic data are inserted into the Integrated Hydrologic Forecast System (IHFS) database. The application can be thought of as having three components:

1) A decoder/parser which reads the SHEF-encoded data and translates the data into a general form, where each data value has an associated set of attributes, such as the physical element it represents and the time of the value.
2) A database poster which reads this general form and writes the data value to the proper tables within the IHFS database.
3) A controlling function (i.e. driver) that controls the invocation and sequencing of these two activities.

This document describes the operational aspects of the poster and the driver operations of the SHEF Decoder implemented with the IHFS database; it does not discuss other versions of the SHEF Decoder which post their data to other destinations. Also, it does not discuss the actual parsing and decoding of the SHEF-encoded data. That is described in detail in the National Weather Service Manual 10-944, dated September 17, 2002. Manual 10-944 supercedes the previous SHEF format contained in the WHFS Standard Hydrometeorological Exchange Format (SHEF) Version 1.3 manual, dated March 1998.

First, a brief description of the input and output data sets is given in Section 2. The input data includes a set of switches and options in the form of application token variables. Each of the tokens are described in Section 3. The overall program processing is described in Section 4, while the detailed processing performed on each product is described in Section 5. Both the overall and product processing are controlled in part by the token settings described in Section 3.

The AWIPS version of the SHEF Decoder that this document applies to is AWIPS Release OB1. The previous version of this document, dated September 8, 2000, applied to Release 5.0. Most of the features described herein also apply to releases prior to Release 5.0; the changes that were implemented for Release 5.0 are described in Section 6.

2. INPUT/OUTPUT DATA

The following data sets serve as input to the SHEF Decoder application:

1) SHEF-Encoded Products - Text files containing the actual SHEF-encoded data to be decoded, processed, and posted to the database. These files are contained in the directory defined by the application token *shef_data_dir* (tokens are discussed below). A special file, called a stop file, may also be located in this same directory. When present, this file directs the application to cease execution.

2) SHEF Application Settings - A collection of tokens and their settings, which are defined in text files referred to as application defaults (Apps_defaults) files. These token values are expected to change rarely once they are configured for a particular office. Tokens are described in a later section of this document.

3) SHEF Parameters - A text file containing the recognized values of the SHEF attributes associated with each value; e.g. the allowable SHEF physical element codes. This file is provided with the application and is not expected to change. Its name is SHEFPARM and it is contained in the directory defined by the application token *shefdecode_input*.

4) IHFS Database - The IHFS database is a relational database implemented in the Informix database environment provided on AWIPS workstations. The database contains assorted switches associated with a given station or area that control how the decoded SHEF information should be processed by the SHEF Decoder.

The following data sets are generated by the SHEF Decoder application:

1) IHFS Database - The IHFS database receives the decoded data and represents the primary output of the SHEF Decoder. The data are organized in the IHFS database based on the SHEF attributes of the data, i.e. the SHEF physical element, duration, type-source, duration, and extremum values, in addition to the station identifier and the time of the data, and possibly the forecast issuance time.

2) Decoded Output - A binary file, for temporary use, that contains the decoded general form of the SHEF data file. It is generated by the parser component of the application and is read by the posting component. After each product's data are posted, this file is removed. This file is named SHEFOUT, and is stored in the same location as the SHEF-encoded data files.

3) Daily Program Logs - A text file containing a running daily log that summarizes the application processing. For each product, about a dozen lines are written to the log file which summarize the product processing. Also, whenever the application is started, it writes the values of the control settings it has read to the log file. The log files are named shef_decode_log_MMDD, where MMDD is the month-day, and are contained in the directory defined by the application token *shef_decode_log*.

4) Product Logs - A text file containing a log that provides a detailed summary of the processing of a single product. This log contains a copy of the input SHEF-encoded data; any errors that may have occurred during the parsing process are written immediately after the line in the product that caused the error. It also includes information regarding the posting operations performed on the data and the same summary information that is logged to the daily log file. The log files are named PRODUCTID.MMDD.HHMMSS, where PRODUCTID is the product id and MMDD and HHMMSS, are the month-day, and hour-minute-seconds of the product as read from the product header. These files are contained in the directory defined by the application token *shef_error_dir*.

3. APPLICATION CONTROLS

The SHEF Decoder application uses a set of application controls, referred to as tokens, to control the processing within the application. For each token, the program looks in up to four "places" to read the value of the token. The four places are the environment variable domain of the operating system shell and three sets of files, which contain a set of tokens and their values, and are referred to as application defaults files. The name and location of these three application defaults files are themselves defined by environment variables specified in the application's start script. The SHEF Decoder application, like all applications that use tokens, determines each token's value by looking in four places in the following hierarchal fashion.

1) First, a check is made to see if an environment variable matching the token name is defined in the shell environment. If so, then this gives the token's value.
2) If not defined, the file defined by the environment variable APPS_DEFAULTS_USER is searched. If the file contains the token, then its value is used.
3) If not defined, then the file defined by APPS_DEFAULTS_SITE is searched. If the file contains the token, then its value is used.
4) If not defined, the file defined by APPS_DEFAULTS, which represents the nationally defined token values, is searched. If the file contains the token, then its value is used.

If the token value is still not found, then program uses a default value defined internally, if appropriate.

A description of each of the tokens used within the SHEF Decoder application is given below. The tokens are grouped by their general functional category. The default value shown is the program default; it is NOT the value specified in the national application tokens file! For a detailed presentation of how many of these tokens impact the processing in the SHEF Decoder, refer to Section 4.0 and Section 5.0.

Note that some of these tokens play a significant role in the speed of the SHEF Decoder. Specifically, the settings of the duplicate data processing tokens and posting destination tokens can greatly affect the performance of the application by possibly requiring more data to be posted than is necessary. For these tokens, consideration should be made when setting the token value. Where appropriate, a brief mention of performance impacts is given with the token description below.

Database tokens:

db_name

Name of the database to which the shefdecode poster will write data. The name has the form: hd#_#xxx, where #_# is the database version number, and xxx is the office identifier.
Default = N/A

server_name

Name of the database server, typically set to ONLINE.
Default = N/A

Directory location tokens:

shefdecode_input

Directory location of input SHEF parameter file. This is normally defined as /awips/hydroapps/shefdecode/input
Default = N/A

shef_data_dir

Directory location of input SHEF-encoded products. This directory may also contain the special stop file. The location is normally defined as /data/fxa/ispan/hydro
DEFAULT = N/A

shefdecode_log

Directory location of the daily log files, which is normally set to /awips/hydroapps/shefdecode/logs/decoder
Default = N/A

shef_error_dir

Directory location of the product log files, which is normally set to /awips/hydroapps/shefdecode/logs/product
Default = N/A

Logging tokens:

shef_keeperror

Controls the dispensation of the product log files.

ALWAYS = Keep product log files always.
IF_ERROR = Keep product log files only when errors or warnings occur.
Default = ALWAYS

dupmess

Specifies whether to log messages in the product log files about duplicate data, which can occur if a value is sent for a location, time, etc. for which a value already exists.

ON = Log messages about duplicate data.
OFF = Don't log messages about duplicate data.
Default = ON

locmess

Specifies whether to write messages in the product log file about stations and areas not defined as either a location or as an area, such as a basin, county, or zone.

ON = Log messages about undefined locations.
OFF = Don't log messages about undefined locations.
Default = ON

elgmess

Specifies whether to write messages in the product log files about the posting "eligibility" of a known location's value. The "eligibility" refers to whether the specific type of data for the given location's data should be posted.

ON = Log messages about the station eligibility not being satisfied.
OFF = Don't log messages about the station eligibility.
Default = ON

shef_perflog

Controls whether the performance logging feature is enabled. When enabled, the decoder will create a separate log file that tracks the timing of selected operations within the decoder for the purpose of monitoring performance. The information is written to the file shef_perf.log which is located in the directory defined by the token *shefdecode_log*. This feature should be used only if necessary because the logging of the performance information itself has an effect on the performance. A line is written to the file for each record that is processed. Therefore, this file can grow to be quite large if the application is running for an extended period. When the application restarts, this file is overwritten, so remember to rename the file if later analysis is desired. This feature is intended for use by knowledgeable operators only. Appendix A gives the format of the performance log file.

ON = Enable performance logging.
OFF = Disable performance logging.
Default = OFF

Processing tokens:

shef_sleep

Specifies how long the application should wait, in seconds, after processing all the input SHEF-encoded data files, before looking to see if any new product files have arrived.

Default = 10

Data Time Window tokens:

shef_winpast

Specifies how many days in the past observed data will be accepted. Data for times before this number of days prior are rejected.

Default = 10

shef_winfuture

Specifies the number of minutes in the future, relative to the current time, that time stamp of observed data can have for the data to be posted.

Default = 30

Duplicate Data Processing tokens:

shef_use_revcode

Indicates whether the poster should consider the SHEF revision code when a value has a duplicate record already in the database. The revision code can be encoded as part of the SHEF encoded information, using the ".AR", ".BR", or ".ER" feature of SHEF.

0 = Consider the revision code when deciding whether to overwrite duplicate data. If the revision code is not set on the new data, then the data is not replaced, unless the token *vl_always_overwrite* is set to ON. The token *vl_always_overwrite* is still supported for this release of the SHEF Decoder, although it is considered obsolete.

1 = Do not check the revision code which may be defined for the data; i.e. always overwrite duplicate data. This setting may result in unnecessary database writes, which can slow processing. Note that despite the name of the token, a value of 1, which is typically the value for TRUE, means do NOT use (i.e. ignore) the revision code.

Default = 0

vl_always_overwrite_flag

Specifies whether the application should override the settings of the revision code, and overwrite all data regardless. This token is considered obsolete as the same behavior can be achieved by setting the token *shef_use_revcode* to a value of 1.

ON =	Always overwrite existing data value.
OFF =	Use the shef_use_revcode token setting and the data revision code to control overwrites

Default = OFF

Posting destination tokens:

shef_post_unk

Specifies how data for unknown, i.e. undefined, stations are processed.

NONE =	Do not post any information to the database regarding undefined stations. This setting results in the fastest performance.
IDS_ONLY =	Post only the location identifiers for undefined stations and only store basic information related to the latest product which contained the undefined station.
IDS_AND_DATA =	Post all data from unknown stations. This setting results in the f slowest performance but allows for full monitoring of data from undefined stations.

Default = NONE

shef_load_ingest

Specifies whether the application will automatically create a record in the IngestFilter table containing the station-PEDTSE combinations of entries considered for processing. To be considered by the data poster component of SHEF Decoder, first it checks that the station is defined. If it is, then it checks that the SHEF physical-element, duration, type-source, and extremum attributes are defined (PEDTSE) in the IngestFilter table. If it is, then the value is processed. If not, then this token value can be used to automatically have the location-PEDTSE entry created, and have the station considered by the data poster.

ON =	Load the location-PEDTSE entry to the IngestFilter if needed.
OFF =	Don't load the entry to the IngestFilter.

Default = ON

shef_storetext

Specifies whether the raw SHEF-encoded product should be written to the TextProduct table, which keeps only the latest number of products for a given product identifier, as controlled by the user.

ON = Post raw encoded SHEF text products, which can be reviewed later.

OFF = Don't post the products. This setting results in the fastest performance.

Default = OFF

shef_post_baddata

Specifies how data which fails the certainty quality control checks should be posted. This control settings only effects data which has failed the certainty quality control check, not data which are tagged as questionable. A data which has failed a "certainty" check is considered to be "bad" with certainty, and some offices may prefer that this data be separated from data that is not considered bad. Data fails the certainty check if it fails the gross range check or if the SHEF qualifier code is set to R (rejected) or B (bad).

REJECT = Post failed data to the RejectedData table. This has the effect of removing the data from future consideration, although rejected data can be manually returned to the applicable physical element tables.

PE = Post failed data to appropriate physical element table. This results in the data being co-mingled with the valid data, although its quality code is still marked as bad.

Default = PE

shef_procobs

Specifies whether SHEF "processed" data is treated as observed data. SHEF processed data refers to data which has the first letter of the SHEF type-source code set to "P".

ON = Post SHEF processed data values to the observation physical element data tables and treat them in every way like they are observed data.

OFF = Post to the ProcValue table, which is a table dedicated to storing SHEF processed data only. SHEF processed data are not further stored according to their physical element, as is the case with SHEF observed data.

Default = OFF

shef_post_latest

Specifies whether to check each observed value, and if it is the latest value for the given location and data attributes, store the value to the LatestObsvalue table.

ON =	Post data to the LatestObsValue table, even if it failed the certainty quality control check.
VALID_ONLY =	Post data to the Latest ObsValue only if the value passes the certainty quality control check.
OFF =	Don't post data to the Latest ObsValue table. This setting results in the fastest performance.

Default = OFF

shef_post_link

Specifies whether to store information in the ProductLink table noting that the given location was contained within the associated product. The value and its associated data are not stored in the table, only information that denotes the linkage between the location and the particular product instance.

ON = Post data to the ProductLink table.
OFF = Don't post data to the ProductLink table. This setting results in the fastest performance.
Default = ON

shef_alertalarm

Specifies whether the program should check whether the values exceed pre-defined alert and alarm levels. Only single values are checked against the threshold values; no rate-of-change checking is performed.

ON = Perform alert and alarm checking on the data
OFF = Don't perform alert and alarm checking. This setting results in the fastest performance.
Default = OFF

shef_load_maxfcst

Specifies whether the program should update the RiverStatus table with the maximum forecast data at the conclusion of processing a product that contained at least one forecast stage or discharge value. This information is used in WHFS applications to monitor river conditions.

ON = Update the RiverStatus table.
OFF = Don't update the RiverStatus table. This setting results in the fastest performance.
Default = OFF

4. PROGRAM PROCESSING

This section summarizes the high-level operations of the SHEF Decoder, described in a sequential manner. The operations are listed in an ordered fashion.

1) Retrieve values of the application environment variables from the .Apps_defaults file(s). The token values control major aspects of the data posting process and also control the impact on the SHEF Decoder operations is also discussed later. A list of these tokens is given earlier in Section 3., with a description and their default value, if one exists.

2) Open the SHEF parameter input file. This file contains information used by the parser to identify the valid SHEF attribute codes. It is named SHEFPARM and is located in the directory specified by the token *shefdecode_input*.

3) Open the SHEF daily log file. This file is located in the directory specified by the token *shefdecode_log*.

4) Open the Informix database. This opens a database on a database server, both of which are specified in the .Apps_defaults file by the token *db_name* and *server_name*, respectively. The database is opened once and is expected to remain accessible by the application.

5) Check for any SHEF encoded files in the input directory defined by the token *shef_data_dir*. Any file in the directory, except for a few special named files, are assumed to be SHEF encoded. The files in the input directory need not follow any special naming convention. Only regular files are considered, i.e. directory files are not considered.

   When assembling the list, certain files are ignored, namely the SHEF parser output file (SHEFOUT), the SHEF stop file (stop_shefdecode), the file list file (files.list), which contains the list of files to consider from the previous directory query, and the SHEF process identifier file (shef_pid.dat), which is used by the start script to try and prevent multiple instances of the SHEF decoder from operating on the same directory.

   A list of files to process is assembled. For each item in this list, the following processing occurs.

   5.1) Open the input file and open the SHEFOUT output file. If either open fails, discontinue processing on the file.

   5.2) Read the header information in the SHEF input file. This includes the product identifier and the product date and time. If the product identifier is missing, the value of MSGPRODID is assigned. The date and time read

from the product gives only the day-of-the-month, and the hour and minute. The year and month are assigned from the system clock. If the date and time cannot be read from the product, the values from the system clock are assigned.

5.3) Open the product log file associated with the particular product. The name of this log file is based on the product identifier and time and is located in the directory specified by the token *shef_error_dir*.

5.4) Parse the encoded information in the SHEF input file. The decoded information is written to the SHEFOUT output file.

5.5) The first time the parser is invoked, read then close the SHEF parameter input file. This task is only performed once even though it is an item within Step 5, and Step 5 is repeated.

5.6) Read the decoded information that is stored in the SHEFOUT file and post the information into Informix database. The posting process follows a detailed sequence of steps that are described in Section 5.

5.7) Close and remove the SHEF input file.

5.8) Close and remove the SHEFOUT output file.

5.9) Close the product log output file.

5.10) Check for existence of a stop file. If one exists then abort application. This closes the database and daily log output file.

Repeat step 5) operations to process any additional SHEF input files.

6) Check for existence of a stop file. If one exists then abort application. This closes the database and the daily log output file.

7) Suspend program execution for duration specified by the token *shef_sleep*. After this pause, continue.

8) Check the current time and if the date has changed in reference to the date of the daily log file, then close the existing log file, and open a new daily log file.

Return to step 5) and processes any files in the input directory. Repeat steps 5-8 indefinitely.

The daily log files and the product log files are eventually purged from the file system by

a purging process that is scheduled to run on a regular basis. In the WFO Hydrologic Forecast System (WHFS) implementation, this task is part of the purges performed by the purge_files script, which typically runs every 4 hours using the UNIX crontab feature.

The IHFS data tables that contain the large volume of data posted by the SHEF Decoder are purged by the db_purge application, which typically runs every 24 hours, and deletes all data older than a specified time from the appropriate tables.

The SHEF Decoder application is started using a start script, called start_shefdecode. This script is located in the directory: /awips/hydroapps/shefdecode/bin. The shefdecode is automatically started when the AWIPS data ingest processes are started, and is expected to run continuously. After times when database maintenance operations are performed, the start script is used to restart the SHEF Decoder application. Note that the start script prevents non-designated users from starting the application. Typically, the application can only be started by the user “oper”.

When the SHEF Decoder first starts, it creates a file called shef_pid.dat in the input directory defined by the token *shef_data_dir* and which contains the process id of the application. This file is read by the start script and the script checks if there is a process currently running that matches the process identifier read in the file. If so, then the SHEF Decoder is considered to be currently running, so a new instance of the SHEF Decoder is not started. If not, then the process id file is assumed to contain a process id that was for a terminated instance of the SHEF Decoder, so the new instance is permitted to be started. Note that a limitation of this method is that it will only look on the current machine for a process id match; multiple instances can be invoked if they are running on different machines. This is a very unstable situation for the SHEF Decoder and should not be permitted under any circumstances. The classic symptom of having two decoders processing the same input directory is the occurrence of numerous file open, file close, and file delete errors.

To stop the SHEF Decoder application, the stop_shefdecode script, located in the same directory, can be used. Note that if the decoder is processing a file when the stop script is invoked, it will finish processing the file before shutting down. If the file being processed is large, this may take a short moment before the program is actually stopped. Stopping the SHEF Decoder application is necessary before performing certain database maintenance operations. It is generally only done by system administrators or knowledgeable operators. As with the start script, the stop script prevents non-designated users from stopping the application.

5.0 SHEF DATA POSTING PROCEDURE

Each product, and the individual data elements within each product, is processed in a manner discussed in this section. Most of this discussion is in the form of a detailed four-page logic diagram given in Appendix A. Although the brief summary below is informative, only the diagrams provide the complete detail necessary to appreciate all the aspects of the posting procedure.

In summary, a product has two components:

1) the header information which applies to the product as a whole, and
2) the multiple, individual data records which are each for a given location, valid time, and SHEF attributes including the physical element, duration, type-source, extremum. In the case of forecast data, there is also a forecast basis time and probability code. These SHEF attributes uniquely define the data record and serve as the "key" for the record in the physical element database tables.

Information derived from the product header, such as the product identifier, are associated with each record posted into the physical elements tables. Product header information can also be stored independent of the data records, such as storing the text product itself, and storing information about the time the product was last received. The number of versions to keep for a given product identifier is specified in the PurgeProduct table, along with the time the product was last received. If the number of versions to keep for an identifier is greater than zero, then the product is posted to the TextProduct table and any older versions are purged as needed to limit the number retained to the specified number.

After the product header based information is processed, the individual data records are then processed. Each individual data record is checked to determine whether it should be posted to the database. Data for undefined stations (or areas) can be ignored, or posted in an abbreviated form, or can be posted in their entirety. If the station is defined but the specific SHEF attributes for the station are not defined, the data is similarly ignored, unless the token *shef_load_ingest* instructs the SHEF Decoder to permanently define and recognize the SHEF attributes, in which case the current data are posted. Alternatively, a station can be defined, but in a way that the posting of its data is explicitly turned off.

Assuming that the data record is to be posted, the data record is checked in multiple ways. First, a set of adjustment factors may be specified for the data key of the record; the value is adjusted numerically if there is a match. Then the value is checked for quality control purposes, and if instructed, can be checked for alert/alarm purposes. The data are then posted to the appropriate table associated with the SHEF physical element and type-source code of the data. The type-source indicates whether the data is for an observation, forecast, or other type of data. The quality control operations of the IHFS data processing, including those operations performed by the SHEF Decoder,

are described in a separate document.

If the value is a duplicate value, user specified token values control how the duplicate value is handled. Also, if the value fails certain quality control tests, then other user instructions control how the value is posted. If the user instructs alert/alarm checking to be performed, then if the value exceeds alert/alarm thresholds, the data record is posted to a table containing only the alert and alarm data, in addition to being posted to the physical element tables. User instructions also control whether the latest observed data is posted.

Posting data to certain tables can result in subsequent operations performed on the data value by procedures defined within the Informix database definition, outside of the SHEF Decoder. Informix procedures are initiated by Informix triggers specified for observed height, discharge, and precipitation data. The trigger “triggers” the procedure anytime a record is inserted or updated to one of these tables. There is no token available for turning these triggers off.

If the product contains forecast height or discharge data, then after all the records in the product are processed, the token *shef_load_maxfcst* may instruct the SHEF Decoder to perform some post-processing to determine the maximum values for the station forecast data.

Throughout the entire posting process diagramed on the following pages, errors can occur which may result in messages written to the product log files. To avoid unnecessary clutter, the linkages between the log files and respective processes are not show in the diagram. Also, often when a value is written to a table, it checks for an existing duplicate. The linkage between the table and the process which checks for the duplicate is not shown in order to reduce clutter. Only the linkage indicating the actual insert or update of a value to a table is shown.

Additional information on the IHFS database structure, including a data dictionary and assorted entity-relationship diagrams are also available as separate documents. The data flow diagram for the SHEF Decoder is a part of this separate documentation. To allow this document to be as complete as possible, it is included in Appendix B.

6.0 NEW FEATURES SINCE AWIPS RELEASE 5.0

Release 5.1.1 changes:

1. Removed ObsValue, ObsValueDup, FcstValue tables.
2. Modified performance logging to account for removed tables.

Release 5.1.2 changes:

1. Modify the poster to handle paired/vector data such as TB, TV, and NO data. To accomplish this, added new table, PairedValue, to store the observations for the following special SHEF physical elements:

HQ (distance to river's edge from stake)
MS (soil moisture at various depths)
NO (gate opening)
ST (snow temperature at various depths)
TB (temperature at depth under bare soil)
TE (air temperature at various elevations)
TV (temperature at depth under vegetated soil)

These physical elements are special in that they require an additional independent variable for the primary key; this independent variable is required because these data are composed of several values at the same station, for the same physical element, and for the same time; for the NO physical element we get gate number and gate opening; for the TB and TV physical elements we get soil depth and soil temperature; for the HQ physical element we get stake number and distance to river; for the MS physical element we get soil depth and soil moisture; for the NEW ST physical element we get depth in the snow pack measured from the ground and snow temperature; for the new TE physical element we get elevation above the ground and the air temperature; the structure is similar to all other dynamic forecast PE data tables such as FcstHeight in order to make this new table totally general in case some new forecast data types appear in the future that are structured like this; the structure is: "lid", "pe", "dur", "ts", "extremum", "probability", "validtime", "basistime", "ref_value", "value", "shef_qual_code", "quality_code", "revision", "product_id", "producttime", and "postingtime"; the primary key is composed of the first 9 columns; the only new column (apart from a standard forecast data table) is "ref_value" defined as an Informix integer;

2. Modified so that it includes the number of parsing warnings and errors; request was to also add other fields. Modified so posting time is defined on a per-record basis, not a per-product basis.

3. Added FcstOther table to store forecast data that is not Forecast Height, Temperature, Discharge, or Precipitation.

4. Added new table, Power, to store the observations for the SHEF physical elements VB through VW, generation and generator data; the structure is identical to all other dynamic observation PE data tables such as Height;

5. Added new table, WaterQuality, to store the observations for the SHEF physical elements WA through WV, water quality; the structure is identical to all other dynamic observation PE data tables such as Height;

6. Added new table, FishCount, to store the observations for the SHEF physical elements FA through FZ, fish counts; the structure is identical to all other dynamic observation PE data tables such as Height;

7. Added new fields to performance log file for error counts.

Release 5.2.1 Changes:

1. Added the physical elements (PEs) MD, MN, MV to the list of Pes that are treated as vector/paired values.

Release OB1 Changes:

1. Added data adjustment feature, which allows the shefdecoder to use any locally specified numerical adjustment factors, which are managed via a new HydroBase GUI under Data Ingest/Adjustment Factors. This GUI provides a way to enter, update and delete Data Adjustment Factors which are stored in the IHFS AdjustFactor table.

Note that this table is only read in at the time SHEFdecoder is started and any changes made to this table after the shefdecoder is started are not used in the adjustment of the raw SHEF values. The user will need to stop and start the shefdecoder to have any changes made to the AdjustFactor table to take affect.

The Adjustment Factor logic is as follows. Any SHEF message processed by the new shefdecoder which matches (location ID, Physical Element, Duration, Type Source and Extremum) an entry in the AdjustFactor table will be processed using the following formula and the raw SHEF value will be "adjusted" to create an adjusted value which is then posted into the IHFS database:

Adjusted Value = (((Raw Value / Divisor) + Base) * Multiplier) + Adder

To use this logic you will need to create an entry in the AdjustFactor table using the HydroBase GUI mentioned above, then stop and start the shefdecoder so that this entry is read in by shefdecode.

APPENDIX A. SHEF DECODER POSTING LOGIC DIAGRAM

APPENDIX B. SHEF DECODER DATA FLOW DIAGRAM

APPENDIX C. FORMAT OF PERFORMANCE LOG FILE

The performance log file feature is described in the discussion of the token *shef_perflog*. This appendix describes the format of the file. The performance log contains one line per product and is designed to be both machine readable (it uses comma-separated value [CSV] format) and human readable (it has character field descriptors strategically placed within the record). Each record has the following 35 fields:

1. product identifier
2. product time - in mmddhhmmss format
3. number of records - total number of records in product
4. posting time - total clock time spent posting

---

5. EW - this literal string identifies the next four fields as being the number of errors and warnings in the parser and poster components
6. Number of parser errors
7. Number of parser warnings
8. Number of poster errors
9. Number of poster warnings

---

10. LGI - this literal string identifies the next three fields as being the time spent accessing the Location, GeoArea, and IngestFilter tables, respectively.
11. Location table - access time
12. GeoArea table - access time
13. IngestFilter table - access time

---

14. LK - this literal string identifies the next four fields as two sets of two values each, where the first number is for the LatestObsValue table and the second is for the ProductLink table
15. LatestObsValue table - number of records processed
16. ProductLink table - number of records processed
17. LatestObsValue table - access time
18. ProductLink table - access time

---

19. HPOF - this literal string identifies the next eight fields as two sets of four values each, where the four values are for the Height, Precip, Other PE tables (e.g. Discharge, Temperature, etc.), and Forecast PE (i.e. FcstHeight, FcstDischarge, FcstPrecip, FcstTemperature) tables, respectively

20. Height table -	number of records processed
21. Precip table -	number of records processed
22. Other observed PE tables -	number of records processed
23. Forecast PE tables -	number of records processed
24. Height table -	access time
25. Precip table -	access time
26. Other observed PE tables -	access time
27. Forecast PE tables -	access time
---	
28. U -	this literal string identifies the next two fields as a pair of a count value, followed by the elapsed time, for processing unknown data
29. UnkStn/UnkStnValue -	number of records processed, for the applicable table
30. UnkStn/UnkStnValue -	access time
---	
31. NP -	this literal string identifies the next four fields as a set of four count values
32. Not posted	count of records instructed to not post
33. Unknowns not posted	count of records not posted because they are unknown
34. Outside time window	count of records not posted because they are outside the allowable time window
35. Ignore duplicate	count of records not posted because they are duplicates and did not meet certain overwrite criteria

All times are given as elapsed time in seconds. A sample log record (ignore the word wrap) is:

KWOHRRSOUN,1227162126,45,1.411,EW,0,0,33,12,LGI,0.092,0.000,0.085,LK,33,9,0.590,0.058,HPOF,5,0,0,0,0.000,0.05,0.000,0.000,U,3,0.017,NP,0,0,0,0

In this example with 45 total records, the total time was 1.411 seconds. The filtering operation (Location and IngestFilter) took 0.178 (=0.092+0.085) seconds, posting to LatestObsValue and ProductLink took 0.59 and 0.058 seconds, posting to the PE tables took 0.05 seconds (all for the height table), and the UnkStn table took .017 seconds.

The CSV format of the file facilitates the development of scripts that can analyze the data and provide summary information. These scripts can be written to interpret the data in an almost unlimited number of ways. Below is an awk script that reads a given performance log file and for each product that is processed, writes the product identifier, the number of records in the product, the time spent processing the product, and the processing rate given in units of seconds per record.

```
#!/bin/awk  -f
BEGIN {printf("   PRODUCT-ID   NUM  TIME  TIM/NUM\n");
sum3=0;
sum4=0
prods=0
FS=","}
{prods++
sum3=sum3+$3;
sum4=sum4+$4;
printf("%15s %4d %6.2f %5.3f\n", $1, $3, $4, $4/$3) }
END { print "Total Records=", sum3,  "Total Time=",sum4, "Total Products=",prods;
      print "Avg. sec/prod=",sum4/prods,"Avg sec/rec="sum4/sum3}
```